SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Evaluating learning algorithms: An efficient way/Efficient ways to find Regular Inductive Statements

Van Tu Nguyen

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Evaluating learning algorithms: An efficient way/Efficient ways to find Regular Inductive Statements

Titel der Abschlussarbeit

Author:	Van Tu Nguyen
Supervisor:	Prof. Dr. Dr. h. c. Javier Esparza
Advisor:	Christoph Welzel-Mohr
Submission Date:	Submission date

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, Submission date Van Tu Nguyen

Abstract

Regular Model Checking is a widely used paradigm for verifying parameterized and infinite-state systems that occur naturally when a program uses queries, stacks or intergers, and so on. One of the major challenges in verifying parameterized and infinite-state systems is determining the set of states that can be reached from a set of initial states, which is an undecidable problem and necessary incomplete.

To address this issue, we propose a semi-decision procedure for regular model checking that uses inductive statements to over-approximate all reachable states. A statement is considered inductive if it only relates a state satisfying ϕ with states that also satisfy ϕ. We demonstrate how the statements are encoded and their *interpretations* defined, which is crucial for understanding the encoded statements.

We discuss the primary mechanism of learning automata, which involves the use of membership queries and equivalent queries. During the learning process, the Teacher and the Learner interact with each other. The Teacher has knowledge of the target language, while the Learner has the opportunity to ask two types of queries: membership and equivalence queries. Additionally, we will cover four active learning algorithms: L^*, NL^*, Kearns-Vazirani, Rivest-Schapire.

We evaluated the performance of our tool, dodo-cpp, on a set of common examples for parameterized verification, and compared the results of these algorithms. The main findings of this thesis show that: (1.) ..., (2.) ..., (3.) ... (NEED MORE INFO FROM EXPERIMENT)

Acknowledgments

I could not have undertaken this journey without the support of Christoph Welzel-Mohr. I want to thank Christoph Welzel-Mohr for guiding me during the thesis.

I'd like to acknowledge to Chair of Theoretical Computer Science for providing the foundational knowledge necessary for me to finish this thesis.

Last but not least, thanks also go to my family, which was both supportive and patient.

Danke!
Thank you!
Cam on!

Contents

1. Introduction

Computer systems grow in size and permeate more and more areas of our lives, such as PCs, mobile phone, ATMs, car's control systems, an so on. However, as systems become more complex, they become more difficult to protect against mistakes or attacks. Additionally, bugs have the potential to cause significant economic damage, and in some cases, even pose a threat to human lives, such as Pentium-Bug [Kni], Toyota's unintended acceleration [Koo14], etc. Thus the reliability and stability of the software verification are of major importance. Although simulation and testing can detect bugs, it cannot guarantee their absence. Furthermore, in reactive systems like operating systems, servers, and ATMs, when no function is being computed, termination is usually undesirable. For this reason, we are interested in *property checking* or *model checking*.

Model Checking can take on various forms, and one of these is Regular Model Checking (RMC) [Bou+00], where finite automata are used to represent the program that is being verified. It has been widely used in various real-world applications, ranging from adaptive model checking to solving real-world problems ([BFL04], [Abd+04]). RMC is a technique commonly employed to ensure the safety of a system by verifying safety properties. To be more specific, our goal is to confirm that a given program cannot execute in a way that starts from a set of initial configurations and leads to a set of bad configurations. In other words, these bad configurations should be not reached during the program's execution. However, this is an undecidable question in general and tools for Regular Model Checking are necessarily incomplete. A semi-decision procedure was proposed in [Wel23] that utilizes *inductive statements* to ensure that no undesired configuration can be reached from any initial configuration. This means there is at least one inductive statement that is satisfied by the initial configuration but not the undesired one.

Over the past decade, there has been a significant increase in the study of automata learning. This field has produced numerous successful applications, such as computational biology, data mining, robotics, automatic verification, and even the analysis of music. For an extensive survey, please refer to [De 05]. This thesis is inspired by successful automata learning techniques and aims to expand its application to verify safety properties for RMC. One type of automata learning is active learning, in which the learner attempts to acquire a regular language from a teacher

who has complete knowledge of the target language. The learner interacts with the teacher via two type of queries: membership queries and equivalence queries. The first question pertains to whether a word is part of the target language, while the second question is whether a hypothesized automaton can recognize the target language. Once we know how to implement the Teacher to answer the oracles, applying different learning algorithms to obtain a set of inductive statements that are capable of establishing a given safety property is now simple.

The purpose of this thesis is to employ various algorithms such as L^*, NL^*, Kearns-Vazirani, Rivest-Schapire to learn the set of inductive statements of a system. Additionally, we gather and analyze empirical data on the performance of these learning algorithms.

Structure of the thesis

The structure of this thesis is as follows. In Chapter 2 of this thesis, we will establish the notations and definitions that will be used throughout the rest of the paper. In Chapter 3, we will thoroughly explain the *regular transition system* and *inductive statements* as an approach for checking the safety properties of *RMC*. Subsequently, Chapter 4 gives a general introduction to active learning algorithms and their oracles. Furthermore, we will introduce some active learning algorithms that used for our experiments. Chapter 5 will investigate the C++-implemented program to learn a set of inductive statements from systems called *dodo*. Finally, we will summarize and assess our experiment results in Chapter 6 and conclude the thesis with Chapter 7.

2. Preliminaries

In this chapter, we introduce some basic notions and definitions that we use throughout this thesis.

Words and Languages

An *alphabet* Σ is a finite set of symbols. A *word* $u = a_1 \dots a_n$ is a finite sequence of symbols $a_i \in \Sigma$ for $i \in \{1, \dots, n\}$. Σ^* denotes the set of all words over an alphabet Σ. *Regular languages* are those which can be identified by a finite state automaton.

Finite automata

We classify automata into two categories: deterministic and non-deterministic, in order to identify regular languages consisting of finite words.

Definition 2.1: Deterministic finite automaton (DFA)

A DFA is a quintuple $\mathcal{M} = (Q, q_0, \Sigma, \delta, F)$ *where* Q *is a finite set of states with a initial state* $q_0 \in Q$. Σ *is set of alphabet of the automaton. We define the transition* $\delta : Q \times \Sigma \to Q$ *and* F *is a set of final states. Let* $w = a_1 a_2 \dots a_n$ *be a string over the alphabet* Σ. *The automaton* $\mathcal{M}$ *accepts* w *if a sequence of states,* $r_0, r_1, \dots, r_n$ *exist in* Q*:*

- $r_0 = q_0$
- $r_{i+1} = \delta(r_i, a_{i+1}),$ *for* $i = 0, \dots, n-1$
- $r_n \in F$

Definition 2.2: Nondeterministic finite automaton (NFA)

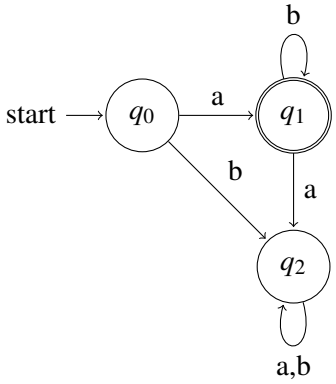
A NFA is a quintuple $\mathcal{N} = (Q, q_0, \Sigma, \Delta, F)$ *where* Q, q_0, Σ *and* F *are defined similarly to Definition 2.1. Let* $w = a_1 a_2 \dots a_n$ *be a string over the alphabet* Σ. *The automaton* $\mathcal{N}$ *accepts* w *if a sequence of states,* $r_0, r_1 \dots, r_n$ *exist in* Q:

- $r_0 = q_0$
- $r_{i+1} \in \Delta(r_i, a_{i+1})$, *for* $i = 0, \dots, n-1$
- $r_n \in F$

Here are two examples of presenting regular languages using DFA and NFA. $\mathcal{L} = ab*$ is a regular language that consists of all words over the alphabet $\Sigma = \{a, b\}$, which begin with one "a" and are followed by either any number of "b" or nothing at all.

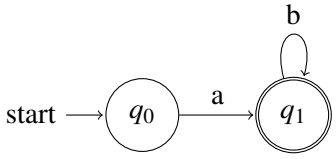
Automaton 2.2: DFA for language $\mathcal{L} = ab*$

Automaton over the alphabet $\Sigma = \{a, b\}$.



Automaton 2.2: NFA for language $\mathcal{L} = ab*$

Automaton over the alphabet $\Sigma = \{a, b\}$.



3. Inductive statements for regular transition system

RMC considers the systems, in which the system's states can be modelled as regular words over a alphabet.

In Section 3.1, we introduce *Regular transition system* (RTS) as a model of RMC to represent the behavior of a parameterized system. Section 3.2 will examine the process of how the statements are encoded.

3.1. Regular transition system

The RMC program model consists of a finite alphabet Σ, a set of initial configurations represented by a regular set over Σ, and regular transition relations between words on Σ^*. For instance, in any parameterized systems $\mathcal{S}$ for linear topology with n processes, the set of finite states for each process is defined as the alphabet Σ. The configurations of this system are presented by the set of words over the alphabet Σ. By setting the length of these configurations to the number of processes, each word represents the corresponding process state. For example, a configuration "$u_1\, u_2\, \ldots\, u_n$" with $u_1, \ldots, u_n \in \Sigma$ indicates that the first process has state u_1, the second process has state u_2, and so on.

Another important point to note is that, in RMC, the system $\mathcal{S}$ defines some semantics, including the regular relation between configurations. In other words, this relation captures the behavior of the system by defining how processes on the system can change states. Formally, we can use a finite-state transducer over $\Sigma \times \Sigma$ to recognize this relation:

Definition 3.1: Transducer

A Σ-Γ-transducer $\mathcal{T}$ is an NFA $\langle Q, Q_0, \Sigma \times \Gamma, \Delta, F \rangle$, we denote for any relations

$$\langle u_1 \dots u_n, v_1 \dots v_n \rangle \in \bigcup_{n \geq 0} \Sigma^n \times \Gamma^n$$

if and only if

$$\langle u_1, v_1 \rangle, \dots, \langle u_n, v_n \rangle \in \mathcal{L}(\mathcal{T})$$

Note that the transducer is length-preserving which means that has the capability to only associate words that have equal length. With $[[\mathcal{T}]]$ as a set of these relations, we define

$$For\ v \in \Sigma^* : target_{\mathcal{T}}(v) = \{u \in \Gamma^* \mid \langle v, u \rangle \in [[\mathcal{T}]]\}$$

As previously mentioned, we will utilize the RTS as a model for the RMC throughout this thesis.

Definition 3.2: Regular transition system (RTS)

An RTS is a triple $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ where

- *Σ is a finite alphabet including all states,*
- *$\mathcal{I}$ is an NFA, which recognizes the language of initial configurations,*
- *$\mathcal{T}$ is a Σ-Σ-transducer of the system, which recognizes the transition relation of the system.*

For every pair of configurations $\langle u, v \rangle$, $u \rightsquigarrow_{\mathcal{T}} v$ implies $\langle u, v \rangle \in [[\mathcal{T}]]$. Moreover, the reflexive transitive closure of $\rightsquigarrow_{\mathcal{T}}$ is denoted with $\rightsquigarrow^*_{\mathcal{T}}$. From any $u \in \mathcal{L}(\mathcal{I})$, we call w is *reachable* in $\mathcal{R}$ if one exists $u \rightsquigarrow^*_{\mathcal{T}} v$. We denote $reach(\mathcal{R}) \in \Sigma^*$ for all such *reachable* configurations.

Example 3.1: *Token passing*

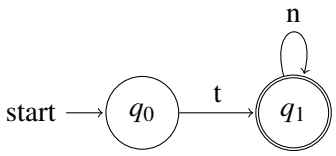
We will provide a simple example to demonstrate how parameterized systems are modelled in *RTS*. The token-passing system is a sequence of agents arranged in a straight line. Each agent is capable of possessing only one token and can transfer it to the adjacent agent on the right. We assume that there is only one agent that can have the token at any given time. At the start, the first

agent holds the token. With each turn, the agent who currently holds the token can pass it to the next agent on the right.

To represent the system as an RTS, we denote the agent that holds the token with "t" and the one that doesn't with "n". Formally, we use the alphabet $\Sigma = \{n, t\}$ for our system. The language $\mathcal{L}(\mathcal{I})$ in Automaton 3.1 defines the set of initial configurations for the *token passing* system as tn*. In other words, the first agent always holds the token, while the following agents do not.

Automaton 3.2: NFA $\mathcal{I}$ for Token passing

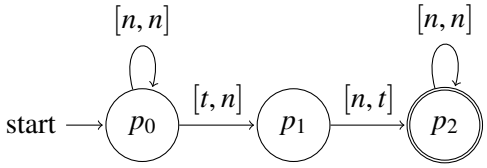
Automaton $\mathcal{I}$ over the alphabet $\Sigma = \{n, t\}$.



Transducer Γ in Automaton 3.2 recognizes the transition relation of the token passing system. Intuitively, the token will be transferred from the left agent to the right agent. Once the token reaches the end of the agents, no further transitions can be made to the configuration.

Automaton 3.2: Transducer Γ for Token passing

Automaton $\mathcal{T}$ over the alphabet $\Sigma \times \Sigma = \{[n,n], [n,t], [t,n], [t,t]\}$.

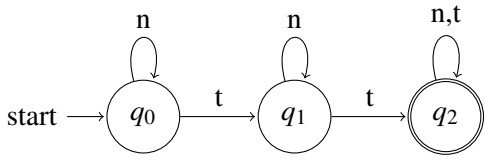


We capture the transitions of the *token passing* system via the language $[n,n]^*[t,n][n,t][n,n]^*$. Consider the following scenario: "n n t n" represents a system with four agents, where the third agent currently holds the token. In the next step, only the third agent can transfer the token to the agent on the right. This can be represented as "n n n t". The system terminates since the last agent holds the token.

We assume that the DFA $\mathcal{B}$ (3.3) represents a set of "unsafe" configurations of the token passing system. In other words, all the configurations that have more than one agent hold the token must not be reached in $\mathcal{S}$. For instance, we must guarantee that $u \not\rightsquigarrow_{\mathcal{T}}^* v$ for all $u \in \mathcal{L}(\mathcal{I})$ and $v \in \mathcal{L}(\mathcal{B})$.

Automaton 3.2: DFA $\mathcal{B}$ for "manytoken"

Automaton $\mathcal{B}$ over the alphabet $\Sigma = \{n, t\}$.



3.2. Inductive statements

In general, we need to determine if a given RTS can produce any undesirable word, which is pre-defined in a regular set. We call this the *Reachability problem*.

Reachability problem

Assume that $\mathcal{L}(\mathcal{B})$ is a set of undesired configurations for a *RTS* $\mathcal{R}$. The question at hand is whether it is possible to reach any undesired configurations from the initial configurations. Formally, we have to compute if $reach(\mathcal{R}) \bigcap \mathcal{L}(\mathcal{B}) = \emptyset$? Because the reachability problem is undecidable [Blo+16], we propose a semi-decision procedure that can provide an answer to the following question: "whether there exists a pair of configurations, v and u, where u satisfies all the inductive statements that v satisfies"?

Indeed, if u is reachable from v, it will satisfy all the inductive statements that v does since they are inductive. By using inductive statements one can ensures that no undesired configurations can be reached from any initial configuration. That means that since a reachable configuration is not part of the undesired language, it satisfies all the inductive statements that the initial does. In simpler terms, at least one inductive statement is satisfied by the initial configuration but not the undesired one.

Encoded statements

We shall now proceed to examine the process of how the statements are encoded. Recall that, the RMC represents the configurations of the system with finite words. In this representation, the i-th word of the configuration indicates the state of the i-th process in the system. The only necessary information required is the index and the state symbol. Formally, we encode the statement information as the function f: $\{1, \ldots, m\} \rightarrow 2^{\Sigma}$, where m is the number of processes

and $f(i) \in \Sigma$ corresponds to the set of possible states of i-th process. For example, we assume a system with the alphabet $\Sigma = \{p, q\}$ and a statement in which all configurations must have a length of 4. Moreover, the first process must have state p or q while the second must have state p. Then, the statement information can be translated to $\{1 \to \{p, q\},\ 2 \to \{p\},\ 3 \to \emptyset,\ 4 \to \emptyset\}$. The function f: $\{1, \ldots, m\} \to 2^{\Sigma}$ is injective, which means that we can uniquely determine a set of states from 2^{Σ} for every index $\{1, \ldots, m\}$. In other words, we can represent the statement as a word with a length of m over the alphabet Σ. For instance, a statement $\{p, q\}$ $\{p\}$ $\emptyset$ $\emptyset$ represents a function $\{1 \to \{p, q\},\ 2 \to \{p\},\ 3 \to \emptyset,\ 4 \to \emptyset\}$.

Without context, these words lack any meaning or explanation. Therefore, we explore the *interpretations* to decide whether the configuration satisfies the statement.

Definition 3.3: Interpretation

For any RTS $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$, we call a pair $\langle \Gamma, \mathcal{V} \rangle$ an Γ-interpretation where Γ is a finite alphabet and $\mathcal{V}$ is a deterministic Σ-Γ-transducer. To express that the pair $\langle u, I \rangle$ belongs to the interpretation of $\mathcal{V}$, we use the notation $u \models I$ and it means that u satisfies I. Formally,

$$\langle u, I \rangle \in [[\mathcal{V}]] \Leftrightarrow u \models I$$

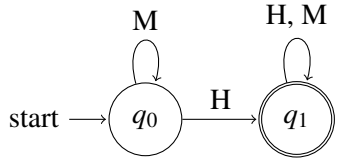
Assume that $u = u_1 \ldots u_n \in \Sigma$ and $I = I_1 \ldots I_n \in 2^{\Sigma}$. The interpretations rely on the satisfaction of atomic propositions $u_i \in I_i$ with $1 \leq i \leq n$ to determine if the configuration u satisfies the statement I. For the sake of clarity, we have outlined three different interpretations: $trap$, $siphon$, $flow$.

Traps In $trap$ interpretation, the method to determine if a configuration u satisfies a statement I involves checking if at least one atomic proposition is true in the proposition. If it is, then it can be concluded that u satisfies the statement I. In other words, if one exits $u_i \in I_i$ for any i with $1 \leq i \leq n$ then $u \models_{\mathcal{V}_{trap}} I$. To illustrate, let's examine whether the configuration *"n n n n"* satisfies the statement {t} {n,t} $\emptyset$ $\emptyset$ by the $trap$ interpretation? We can check this by evaluating the proposition: $n \in \{t\} \vee n \in \{n, t\} \vee n \in \emptyset \vee n \in \emptyset$. Intuitively, we can conclude that u satisfies I or $u \models_{\mathcal{V}_{trap}} I$ since this proposition is true.

We construct the Automaton 3.4 as a Σ-Γ-*transducer* to recognize all pairs of the configuration u and the statement I if $u \models_{\mathcal{V}_{trap}} I$.

Automaton 3.3: DFA for trap intepretation

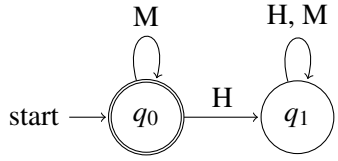
We denote with H all pairs in $\langle \sigma, I \rangle \in \Sigma \times \Gamma$ *such that* $\sigma \in I$ *and M all pairs in* $\langle \sigma, I \rangle \in \Sigma \times \Gamma$ *such that* $\sigma \notin I$.



Siphon Under the siphon interpretation, none of the atomic propositions hold true. To formalize this, we can take into account the proposition: $u_1 \notin I_1 \wedge \cdots \wedge u_n \notin I_n$. When this proposition holds true, we can infer that u satisfies the statement I.

Automaton 3.3: DFA for siphon intepretation

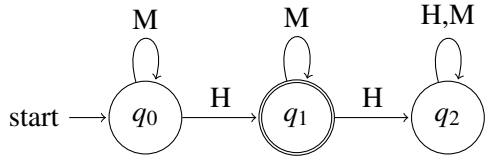
Again, we denote with H all pairs in $\langle \sigma, I \rangle \in \Sigma \times 2^{\Sigma}$ *such that* $\sigma \in I$ *and M all pairs in* $\langle \sigma, I \rangle \in \Sigma \times 2^{\Sigma}$ *such that* $\sigma \notin I$.



Flow This time we want only one atomic proposition is satisfied. For instance, given the configuration $u =$ "n n n t", $v =$ "t t t t" and the statement $I = \{t\}\ \{t\}\ \{t\}\ \{n, t\}$, one can conclude that $u \models_{\mathcal{V}_{flow}} I$ and $v \not\models_{\mathcal{V}_{flow}} I$.

Automaton 3.3: DFA for flow intepretation

We also denote with H all pairs in $\langle \sigma, I \rangle \in \Sigma \times 2^{\Sigma}$ *such that* $\sigma \in I$ *and M all pairs in* $\langle \sigma, I \rangle \in \Sigma \times 2^{\Sigma}$ *such that* $\sigma \notin I$.



Definition 3.4: Inductive statements

Assume that $\mathcal{V}$ *is the* Γ*-interpretation for* $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$, *we define*

$$\begin{aligned} \mathit{Inductive}_{\mathcal{V}}(\mathcal{R}) &= \{I \in \Gamma^* \mid \forall u \rightsquigarrow_{\mathcal{T}} v.\ \mathit{if}\ \langle u, I \rangle \in [[\mathcal{V}]]\ \mathit{then}\ \langle v, I \rangle \in [[\mathcal{V}]]\} \\ &= \{I \in \Gamma^* \mid \forall u \rightsquigarrow_{\mathcal{T}} v.\ \mathit{if}\ u \models I\ \mathit{then}\ v \models_{\mathcal{V}} I\} \end{aligned}$$

Inductively, the statement I will remain satisfied throughout every possible transition of the RTS. Thus, every configuration reachable from any v also satisfies all the statements that v satisfies. Formally, when $v \rightsquigarrow^*_{\mathcal{T}} w$ if $v \models_{\mathcal{V}} I$ then also $w \models_{\mathcal{V}} I$. In other words, we can provide $v \not\rightsquigarrow^*_{\mathcal{T}} u$ if there exists a statement I such that $v \models_{\mathcal{V}} I$ but $u \not\models_{\mathcal{V}} I$.

Example 3.2: *Inductive statements of token-passing system,* $\mathcal{V}_{trap}$.

To illustrate, let us examine the example:

$$\{n\}\ \{n\}\ \{n\}^*\ (\{t\}\ \{n\}\ \{n\}^* \mid \{t\})^1 \subseteq \mathit{Inductive}_{\mathcal{V}_{trap}}(\mathcal{R})$$

For the fixed length of configuration n = 4, all possible inductive statements are

$$\begin{array}{cccc} \{n\} & \{n\} & \{n\} & \{n\}, \\ \{n\} & \{n\} & \{n\} & \{t\}, \\ \{n\} & \{n\} & \{t\} & \{n\} \end{array}$$

We can observe that every possible configuration that can be reached from the initial configuration $u =$ "$t\ n\ n\ n$" also satisfies all the statements that the initial configuration satisfies. Intuitively, a

[1] The regular set has been learned in practice by dodo-cpp using token-passing system with trap interpretation, and the "manytoken" property.

configuration $w =$ "$n\ n\ t\ n$" can be reached from u both satisfy all above statements, which are also satisfied by u. It also concludes that $v =$ "$t\ t\ t\ n$" $\in \mathcal{L}(\mathcal{B})$ can not reached from the initial configuration $u =$ "$t\ n\ n\ n$" $\in \mathcal{L}(\mathcal{I})$ because u satisfies the statement $I = \{n\}\ \{n\}\ \{n\}\ \{t\}$, but v does not.

By this way, we can guarantee that no bad configurations can be reached by checking the satisfaction of the configurations with the statements.

Definition 3.5: Potential reachability

Let $\mathcal{R} = \langle \Sigma, \mathcal{I}, \mathcal{T} \rangle$ be any RTS and $\langle \Gamma, \mathcal{V} \rangle$ any interpretation. We write $v \Rightarrow_{\mathcal{V}} I$ if and only if $u \models_{\mathcal{V}} v$ for all $I \in target_{\mathcal{V}}(u) \bigcap Inductive_{\mathcal{V}}(\mathcal{R})$.

Lemma 3.1:

Let $\mathcal{R} = (\Sigma, I, T)$ be an RTS, $\langle \Gamma, \mathcal{V} \rangle$ an interpretation, and S a NFA over the alphabet Γ. Then there exists a $\Sigma - \Sigma -$ transducer C such that

$$[[C]] = \left\{ \langle u, v \rangle \in \bigcup_{n \geq 0} \Sigma^n \times \Sigma^n \mid \forall I \in \mathcal{L}(S)\ .\ if\ u\ \models_{\mathcal{V}}\ I\ then\ v \models_{\mathcal{V}} I \right\} \tag{3.1}$$

Because regular languages are closed under complement, we can define $\overline{C}$ as followings:

Lemma 3.2:

Let $\mathcal{R} = (\Sigma, I, T)$ be an RTS, $\langle \Gamma, \mathcal{V} \rangle$ an interpretation, and S a NFA over the alphabet Γ. Then there exists a $\Sigma - \Sigma -$ transducer $\overline{C}$ such that

$$[[\overline{C}]] = \left\{ \langle u, v \rangle \in \bigcup_{n \geq 0} \Sigma^n \times \Sigma^n \mid \exists I \in \mathcal{L}(S)\ .\ if\ u\ \models_{\mathcal{V}}\ I\ then\ v \not\models_{\mathcal{V}} I \right\} \tag{3.2}$$

Note that, these Lemmas have been proved in the previous work [Wel23]. We will utilize them later for constructing the equivalent oracle in the implementation Chapter 5.

4. Algorithmic Learning of Finite Automata

Learning automata is a computational model for solving problems, where an agent learns to optimize its behavior by interacting with an unknown environment. The agent, also known as a learner, observes the feedback from the teacher, updates its internal state, and adjusts its actions accordingly. The primary mechanism of learning automata is the exchange of information between the learner and the teacher. In the field of automata learning, there are generally two distinct settings: active and passive learning. Passive algorithms are provided with a fixed set of examples consisting of strings that the target language should either accept or reject. Active algorithms, unlike passive ones, have the ability to add more examples by asking further queries. However, in this thesis, our focus is solely on active learning. We do not introduce passive learning here but refer the interested reader to [AMP22].

This chapter intends to offer a comprehensive insight into the learning automata process, specifically discussing the roles and responsibilities of the Teacher and Learner in Section 4.1. In Section 4.2, we will introduce multiple active algorithms that we will use for our experiment.

4.1. The oracles

In this learning scenario, the teacher is proficient in the language being taught and is responsible for answering the questions posed by the learner. The learner is given the opportunity to ask two types of queries - *membership* and *equivalence*. Membership queries are used to classify a word based on whether it belongs to the language being taught or not. During equivalence queries, we assess whether the assumed conjecture accurately recognizes the target language. If it does, the learning process ends. If it doesn't, we continue the learning process by providing a counter-example for the learner to use in improving the conjecture.

Membership oracle The learner provides a word $w \in \Sigma^*$, the teacher replies "yes" or "no" depending on whether $w \in \mathcal{L}$ or not.

Equivalent oracle The conjecture made by the leaner that is usually presented in the form of a DFA or a NFA M (Table 4.1). The teacher then verifies whether $\mathcal{M}$ accurately represents the

target language $\mathcal{L}$. If $\mathcal{L}(M)$ is equivalent to $\mathcal{L}$ it returns "yes", otherwise return a counterexample $u \in \Sigma^*$ with $u \in \mathcal{L}(\mathcal{M}) \iff u \notin \mathcal{L}$ or $u \in \mathcal{L} \iff u \notin \mathcal{L}(\mathcal{M})$.

On equivalent oracle, the *Teacher* can return a positive counterexample or a negative counterexample [Che+17]. A positive counterexample is a missing word in the conjecture but present in the target. The negative one is defined symmetric.

Active learning faces the challenge that, during the learning process, it is affected by counterexample quality. For example, if a teacher provides an excessively long counterexample, the learner is forced to process the whole word with no alternative. Therefore, it is crucial that the teacher must have a clear and specific understanding of the target language. Since we know how to implement the teacher to answer the oracles, it is now simple to apply different learning algorithms.

4.2. Algorithms

A learning algorithm try to acquire the knowledge of a regular language by asking questions to the teacher. The conjecture can be in form of DFA or NFA depends on the algorithm architecture. Throughout this thesis, we utilize a variety of these algorithms such as Angluin's L* Algorithm, NL* Algorithm, Rivest and Schapire's Algorithm, and Kearns and Vazirani's Algorithm. In table 4.1 illustrates that the majority of algorithms make use of DFA as their automata type, except for the NL* Algorithm, which uses an NFA.

Algorithm	Automata type
Angluin's L* Algorithm	DFA
NL* Algorithm	NFA
Rivest and Schapire's Algorithm	DFA
Kearns and Vazirani's Algorithm	DFA

Table 4.1.: Learning algorithms and it's conjectured automata type.

4.2.1. L*

L* learning automata was introduced by Angluin in 1987 [Ang87], also called Angluin's algorithm. L* has the ability to learn a regular set which is unknown initially, from any *Teachers*. During the process of learning, it stores information in an observation table $O = (S, E, T)$ where

$S \subseteq \Sigma^*$ is a nonempty *prefix-closed* [1] set, a finite *suffix-closed* [2] set E, and T : $(S \cup S \cdot A) \cdot E \rightarrow \{0,1\}$ is a mapping that stores membership information of the table entries. To fill value into the table entries, the algorithm asks membership queries. If the membership queries for a word u return true, we assign $T(u) = 1$. Conversely, if the target language does not accept word u, $T(u) = 0$.

Let's take a closer look at the inner workings of the *Angluin's algorithm*. Assume that $s \in S \cup S \cdot A$, we denote $row(s)$ for finite function f where:

$$f_s : E \rightarrow \{0,1\} \; with \; f_s(e) = T(s \cdot e)$$

The observation table has two properties: *closed* and *consistent*. The set S comprises all the representatives of a language. As a result, all words in the set $(S \cdot A)$ should have the same value when compared to their representatives S. We call that a *closed* property of the table. Formally, $\forall a \in S \cdot A$ there exists that $s \in S$ and $row(a) = row(s)$. We call an observation table is *consistent* when two representatives $s_1, s_2 \in S$ have the same value, then also their suffixes. More precisely, $\forall a \in A$ when $row(s_1) = row(s_2)$ then $row(s_1 \cdot a) = row(s_2 \cdot a)$. After ensuring the table is both *closed* and *consistent*, it becomes possible to create a deterministic finite-state acceptor from the observation table, also referred to as a *conjecture*. To be more specific, *Angluin's algorithm* constructs the conjecture $\mathcal{H} = (Q, q_0, \Sigma, \delta, F)$ where:

$$Q = \{row(s) : s \in S\},$$

$$q_0 = row(\lambda),$$

$$F = \{row(s) : s \in S \; and \; T(s) = 1\},$$

$$\delta(row(s), a) = row(s \cdot a).$$

The Angluin's algorithm is presented in the Algorithm 1 in pseudocode. Essentially, in the begin of learning process, the algorithm guarantees that the table are *closed* and *consistent* by repeatedly modifying the columns and the rows of the table. Following each extension of the table, the algorithm requests membership queries for the table entries, which still has no membership information yet. For all $u \in (R \cup R \cdot \Sigma) \cdot S$, if the *Teacher* replies "yes", then we set $T(u) = 1$, otherwise $T(u) = 0$. After the observation table meets the necessary requirements of being *closed* and *consistent*, the algorithm moves forward with creating a conjecture. This conjecture is then presented to an equivalence query. The learning terminates once the teacher replies "yes" on an equivalence query. In case the teacher presents a counterexample $t \in \Sigma^*$, the algorithm will

[1] A set of strings S is called prefix-closed if: $uv \in S \implies u \in S$

[2] A set of strings S is called suffix-closed if: $uv \in S \implies v \in S$

adjust the table by including it and all of its preceding elements to S, then start over by returning to line 1 (Algorithm 1).

Algorithm 1 Algluin's learning algorithm [Ang87]

Input: A teacher for a regular language $L \subseteq \Sigma^*$

Initialize the observation table (S, E, T)

Ask membership queries for λ and each $a \in \Sigma$

Repeat:

1: **while** (S,E,T) is not closed or not consistent **do**
2: **if** (S,E,T) is not consistent **then**
3: find s_1 and s_2 in S, and $e \in E$ such that
4: $row(s_1) = row(s_2)$ and $T(s_1 \cdot a \cdot e) \neq T(s_2 \cdot a \cdot e)$,
5: add $a \cdot e$ to E,
6: conducts membership queries.
7: **end if**
8: **if** (S,E,T) is not closed **then**
9: find s_1 and $a \in \Sigma$ such that
10: $row(s_1 \cdot a)$ is different from $row(s)$ for all $s \in S$,
11: add $s_1 \cdot a$ to S,
12: conducts membership queries.
13: **end if**
14: **end while**
15: Once (S, E, T) is closed and consistent, make $M = M(S, E, T)$
16: **if** the Teacher replies with a counter-example t, then **then**
17: add t and all its prefixes to S
18: conducts membership queries.
19: **end if**

Util the Teacher replies "yes"

Terminate and return a conjecture $\mathcal{M}$

4.2.2. NL*

In general, a nondeterministic finite automata *NFA* is often preferable to a deterministic finite automata *DFA* due to potentially exponential differences in their sizes [Ozo+05]. Therefore, learning algorithms for nondeterministic finite automata are necessary. In this section, we will

introduce another active learning algorithm called the *NL** algorithm [Bol+09], based on the *Angluin′s* approach. The *NL** concludes a residual finite-state automata (RFSA), a subclass of nondeterministic finite automata was introduced in the seminar work [DLT01].

Technically, it is possible to learn an RFSA instead of a DFA by modifying Angluin's algorithm L^* observation table. The proposed method involves selecting *prime rows*[3] as representatives of the automaton's states, rather than utilizing *all rows* of the table. The proceed of the NL^* learning algorithm is mainly the same with *L**. Similar to L^*, it is also repeatedly checked the *(RFSA)-closed*[3] and *(RFSA)-consistency*[3] properties, once the both properties are fullfill, it can contruct the conjecture and ask the equivalent query to the teacher.

The main difference of $NL*$ and $L*$ is the automata type. $NL*$ represents the conjecture with a type of NFA, particularly a (minimal) canonical RFSA, which is always smaller or equal to the corresponding DFA [DLT01].

4.2.3. Kearns-Vazirani

Another active learning algorithm is introduced in this thesis is *Kearns and Vazirani's* algorithm [KV94]. Kearns and Vazirani's algorithm differs from Angluin's algorithm in that it arranges its information in a structured binary tree. It aims to minimize the number of membership queries by storing only one representative for each L-equivalence class in the tree. The data are stored in two non-empty set $R, S \subseteq \Sigma$, where R is comprised of *representatives*. Note that, each representative represents a equivalent class in the language $\mathcal{L}$. The set S includes *separating words* that are used for ensuring that no two representatives represent the same equivalent classes. More formally, *Kearns-Vazirani's* algorithm utilizes a separating word $v \in S$ for two representatives $u \neq u' \in R$ such that $uv \in L \Leftrightarrow u'v \notin L$ is satisfied.

In this thesis, we will not delve into the organization of the table and the construction of the conjecture automata. However, we refer interested readers to the research paper [Nei14].

4.2.4. Rivest-Schapire

The last algorithm we will introduce in this thesis is *Rivest and Schapire* [RS89]. The *Rivest and Schapire* algorithm aims to minimize the amount of data stored in *Angluin's* table by selecting a single representative from each L-equivalence class. The advantages are storing less data and asking less memberships queries.

[3] *prime row, RFSA-closed, RFSA-consitency* are defined in [Bol+09]

4.3. Libalf: the Automata Learning Framework

Libalf [Bol+10] is an open-source program library that facilitates the learning of finite automata. It supports both for active and passive algorithms but in this thesis we only consider the active algorithms such as L*, NL*, Kearns-Vazirani, and Rivest and Schapire.

5. Implementation

We use automata learning algorithms to solve regular model checking problems and generate inductive statements for the parameterized systems.

5.1. Membership oracle

On a membership oracle, the learner provides a statement and asks the teacher if this statement whether inductive or not. As we described in Definition 3.4, a statement I is *inductive* if, for any transition $v \rightsquigarrow u$ where u satisfies I, u also satisfies the statement. Oone can implement the Membership Oracle by checking the acceptance of $\mathcal{M}$, where $\mathcal{M}$ is an automaton for $\overline{Inductive_{\mathcal{V}}(\mathcal{R})}$ and negating the answer (Algorithm 2).

Because regular languages are closed under it's complementation [Hol+15], we define $\overline{Inductive_{\mathcal{V}}(\mathcal{R})}$:

$$\overline{Inductive_{\mathcal{V}}(\mathcal{R})} = \{I \in \Gamma^* \mid \exists u \rightsquigarrow_{\mathcal{T}} w \,.\, u \models I \text{ and } w \not\models I\} \tag{5.1}$$

Let $\mathcal{T} = \langle P, \Sigma \times \Sigma, \Delta, p_0, E\rangle$ is a transducer and $\mathcal{V} = \langle Q, \Sigma \times \Gamma, \delta, q_0, F\rangle$ is an interpretation. The automaton of $\overline{Inductive_{\mathcal{V}}(\mathcal{R})}$ is defined by $\langle Q \times P \times Q, \Gamma, \triangle, \langle q_o, p_0, q_o\rangle, E \times F \times (Q \setminus F)\rangle$ where

$$\triangle(\langle q_1, p, q_2\rangle, I) = \exists \langle \sigma_1, \sigma_2\rangle \in \Sigma \times \Sigma.\ (\delta(q_1, \langle \sigma_1, I\rangle),\ \Delta(p, \langle \sigma_1, \sigma_2\rangle),\ \delta(q_2, \langle \sigma_2, I\rangle))$$

We call a state $\langle q, p, q'\rangle$ is final if $q \in F \wedge p \in E \wedge q' \notin F$.

For every pair of initial word and its reached word through the transducer. Where the initial word is satified by the statement I, the reached word is not. From 5.1 it can guarantee that all statements, that are acepted by this automaton, are non-inductive.

5.2. Equivalent oracle

When the learner provides a conjecture, the teacher checks if it satisfies the safety property. It it does, the teacher return *true*. Otherwise, the learner receives a counter example and repeates the proocess.

Algorithm 2 Membership oracle

Input: *Statement* $\mathcal{I}$

Output: *True* or *False*

begin

1: $\mathcal{M} \leftarrow getAutomaton(\overline{Inductive_{\mathcal{V}}(\mathcal{R})})$
2: **if** $\mathcal{I} \in \mathcal{L}(\mathcal{M})$ **then**
3: return *false*;
4: **else**
5: return *true*;
6: **end if**

end

Algorithm 3 Equivalent oracle

Input: *Statement* $\mathcal{I}$

Output: *True*, X, or $I \in \Gamma^*$

begin

1: $\mathcal{M} \leftarrow getAutomaton(\overline{Inductive_{\mathcal{V}}(\mathcal{R})})$
2: **if** $\mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{M}) \neq \emptyset$ **then** ▷ Make sure that all statements are inductive
3: return $I \in \mathcal{L}(\mathcal{H}) \cap \mathcal{L}(\mathcal{M})$
4: **end if**
5: $\mathcal{D} \leftarrow getAutomatonFor(\mathcal{L}(\mathcal{I}) \circ \overset{\mathcal{L}(\mathcal{H})}{\Rightarrow} \circ \mathcal{L}(\mathcal{B}))$ ▷ Check safety property
6: **if** $\mathcal{D} = \emptyset$ **then**
7: return True
8: **end if**
9: $\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} \ldots \begin{bmatrix} u_n \\ v_n \end{bmatrix} \leftarrow getWordFrom(\mathcal{L}(\mathcal{D}))$
10: $I = disprove(\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} \ldots \begin{bmatrix} u_n \\ v_n \end{bmatrix})$
11: **if** $I = null$ **then**
12: return X ▷ throw exception when can not disprove
13: **end if**
14: return I

end

Firstly, we ensure the automaton only accepts inductive statements. We intersect the automaton of $\overline{\mathit{Inductive}_{\mathcal{V}}(\mathcal{R})}$ with the hypothesis. If there exists any non-inductive statement in the hypothesis, we return it as counterexample.

Since hypothesis $\mathcal{H}$ does not accept any non-inductive statement, we will check with the safety problems to make sure that the hypothesis strong enough. Intuitively, the automaton $\mathcal{D}$ in Algorithm 2 contains all pairs from initial and bad words, which is induced by the inductive statements $\mathcal{L}(\mathcal{H})$. In other words, the safety property is that the inductive statements should not induce the initial and bad word. We return true and terminates the algorithm if $\mathcal{L}(\mathcal{D}) = \emptyset$. Otherwise we obtain a counterexample $\langle u_1 \dots u_n,\ v_1 \dots v_n \rangle \in \mathcal{L}(\mathcal{D})$. Intuitively, we can see that $\mathcal{D}$ is the intersection of the automaton $[[C]]$ (Lemma 3.1) and $\mathcal{I} \circ \mathcal{B}$. Since computing $[[\overline{C}]]$ (Lemma 3.2) is effectively, we will construct the automaton for $[[\overline{C}]]$ and complement it. Let $S = \langle P, \Gamma, \Delta, p_0, E \rangle$ is a transducer and $\mathcal{V} = \langle Q, \Sigma \times \Gamma, \delta, q_0, F \rangle$ is an interpretation. The automaton of $[[\overline{C}]]$ is defined by $\langle Q \times P \times Q, \Sigma \times \Sigma, \triangle, \langle q_o, p_0, q_o \rangle, E \times F \times (Q \setminus F) \rangle$ where

$$\triangle(\langle q_1, p, q_2 \rangle, \langle \sigma_1, \sigma_2 \rangle) = \exists I \in \Gamma.\ \ (\delta(q_1, \langle \sigma_1, I \rangle),\ \Delta(p, I),\ \delta(q_2, \langle \sigma_2, I \rangle))$$

The states that are accepted by this automaton when each its parts are satified:

$$\begin{aligned} \delta(q_1, \langle \sigma_1, I \rangle) &\in F \\ \Delta(p, I) &\in E \\ \delta(q_2, \langle \sigma_2, I \rangle) &\notin F \end{aligned}$$

5.3. The word problem

We are now trying to locate a counterexample $I \in \mathit{Inductive}_{\mathcal{V}}(\mathcal{R})$ that disproves the given hypothesis. This is done to ensure that $u_1 \dots u_n \models_v I$ and $v_1 \dots v_n \not\models I$, our inductive statements will no longer induce this pair. We can also call I an active counterexample since I is in the target language but was missing in the candicate language. It gives rise to the question whether $I \in \mathit{Inductive}_{\mathcal{V}}(\mathcal{R})$ exists such that $u_1 \dots u_n \models_v I$ and $v_1 \dots v_n \not\models I$. It was previously proven by [Wel23] that this problem is in NP. Moreover, since SAT problem is NP-hard, it can be reduced to SAT.

Flow interpretation In this section, we will extract separating inductive statements using CNF-SAT. The entire formular is a conjunction (AND) of clauses, where each clause is a disjunction (OR) of literals. The idea is, we assign each combination of an alphabet $\sigma \in \Sigma$ and the position a variable. In other words, a pair $\langle \sigma, i \rangle$ is a literal which assigns to a variable. The variables have

two value: *true* and *false*. Therefore, σ is a part of I_i if and only if the model value of the literal $\langle \sigma, i \rangle$ is true.

Intuitively, it generates a set of clauses that ensure that exactly one of the literals evaluate to *true*. Recall that a statement is not inductive if there exists one transition $\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} \dots \begin{bmatrix} u_n \\ v_n \end{bmatrix}$ that is accepted by transducer $\mathcal{T}$ for which holds that $x_1 \dots x_n \models I_1 \dots I_n$ and $y_1 \dots y_n \not\models_{\mathcal{V}_{flow}} I_1 \dots I_n$. Formally, we add these clauses to the formular:

$$ExactlyOne(\bigcup_{1 \le i \le n} \{\langle u_i, i \rangle\}) \tag{5.2}$$

and

$$\neg ExactlyOne(\bigcup_{1 \le i \le n} \{\langle v_i, i \rangle\}) \tag{5.3}$$

Clause 5.2 ensures that there is exactly one $1 \le i \le n$ such that $x_i \in I_i$. On the other hand, clause 5.3 guarantees that either there is no or more than one $1 \le i \le n$ such that $x_i \in I_i$. Semantically, we define a state $\langle l, q, k \rangle \in \{0,1\} \times Q_{\mathcal{T}} \times \{0,1,2\}$ corresponds to the observation that one can reach the state q of $\mathcal{T}$ with a word $\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} \dots \begin{bmatrix} u_n \\ v_n \end{bmatrix}$ such that there are k many indices i where $x_i \in I_i$, on the other hand, there are l many indices j where $y_j \in I_j$. We need to ensure that each pair $[x, y]$ we consider is accepted by the *transducer* $\mathcal{T}$. Additionally, in the final step should not result in the same configuration for both the source and target. To achieve this, we encode the state product as literals and define the proposition formula as follows:

$$\begin{aligned}
&\bigvee_{q_0 \in Q_0^{\mathcal{T}}} \langle\langle 0, q_0, 0\rangle, 0\rangle \wedge \neg \bigvee_{f \in F_{\mathcal{T}}} \langle\langle 1, f, 0\rangle, n\rangle \vee \langle\langle 1, f, 2\rangle, n\rangle \\
&\wedge \bigwedge_{1 \le i \le n,\, \left\langle q, \left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right], p \right\rangle \in \Delta_{\mathcal{T}}} \left(\begin{array}{r}
\langle\langle 0, q, 0\rangle, i\rangle \wedge \langle x, i+1\rangle \wedge \langle y, i+1\rangle \implies \langle\langle 1, p, 1\rangle, i+1\rangle \\
\wedge \langle\langle 0, q, 1\rangle, i\rangle \wedge \langle x, i+1\rangle \wedge \langle y, i+1\rangle \implies \langle\langle 1, p, 2\rangle, i+1\rangle \\
\wedge \langle\langle 0, q, 2\rangle, i\rangle \wedge \langle x, i+1\rangle \wedge \langle y, i+1\rangle \implies \langle\langle 1, p, 2\rangle, i+1\rangle \\
\wedge \bigwedge_{k \in \{0,1\}} \begin{pmatrix} \langle\langle k, q, 0\rangle, i\rangle \wedge \neg\langle x, i+1\rangle \wedge \langle y, i+1\rangle \implies \langle\langle k, p, 1\rangle, i+1\rangle \\ \wedge \langle\langle k, q, 1\rangle, i\rangle \wedge \neg\langle x, i+1\rangle \wedge \langle y, i+1\rangle \implies \langle\langle k, p, 2\rangle, i+1\rangle \\ \wedge \langle\langle k, q, 2\rangle, i\rangle \wedge \neg\langle x, i+1\rangle \wedge \langle y, i+1\rangle \implies \langle\langle k, p, 2\rangle, i+1\rangle \end{pmatrix} \\
\wedge \bigwedge_{l \in \{0,1,2\}} \begin{pmatrix} \langle\langle 0, q, l\rangle, i\rangle \wedge \langle x, i+1\rangle \wedge \neg\langle y, i+1\rangle \implies \langle\langle 1, p, l\rangle, i+1\rangle \\ \wedge \langle\langle 0, q, l\rangle, i\rangle \wedge \langle x, i+1\rangle \wedge \neg\langle y, i+1\rangle \implies \langle\langle 1, p, l\rangle, i+1\rangle \end{pmatrix} \\
\wedge \bigwedge_{k \in \{0,1\}\ l \in \{0,1,2\}} \Big(\langle\langle k, q, l\rangle, i\rangle \wedge \neg\langle x, i+1\rangle \wedge \neg\langle y, i+1\rangle \implies \langle\langle k, p, l\rangle, i+1\rangle \Big)
\end{array} \right)
\end{aligned} \tag{5.4}$$

To solve the (CNF-)SAT problem we need to convert the entire of logical formular 5.2, 5.3 and 5.4 in the CNF form, which can be achieved by applying DeMorgan's laws [Wik23].

A polynomial time algorithm for the word problem for $\mathcal{V}_{trap}$ and $\mathcal{V}_{siphon}$ We focus on $\mathcal{V}_{trap}$ since the arguments for $\mathcal{V}_{siphon}$ are analogous. Pseudocode 4 demonstrates how to find the separating statement for trap interpretation within polynomial time. We begin with the statment $I = \Sigma\backslash\{y_1\}\dots\Sigma\backslash\{y_n\}$. If a transition $\left[\begin{smallmatrix}x_1\\y_1\end{smallmatrix}\right]\dots\left[\begin{smallmatrix}x_n\\y_n\end{smallmatrix}\right]$ exists such that $x_1\dots x_n$ satisfies the current statement and $y_1\dots y_n$ does not, then remove x_i from the i-th letter of the statement for all $1 \leq i \leq n$. To prove that this approach can be computed in polynomial time, refers to [Wel23].

Algorithm 4 Disprove (polymial time algorithm for trap)

Input: $\left[\begin{smallmatrix}x_1\\y_1\end{smallmatrix}\right]\dots\left[\begin{smallmatrix}x_n\\y_n\end{smallmatrix}\right]$ and transducer $\mathcal{T}$

Output: Inductive statement I

begin

for $i = 1;\ i \leq n;\ i = i + 1$ **do**
    $I_i = \Sigma\backslash\{y_i\}$
end for
while $\langle v, I\rangle \in \mathcal{L}(\mathcal{V}_{trap})$ **do**
    if $\exists \left[\begin{smallmatrix}x_1\\y_1\end{smallmatrix}\right]\dots\left[\begin{smallmatrix}x_n\\y_n\end{smallmatrix}\right]$ where $u_1\dots u_n \models I$ and $v_1\dots v_n \not\models I$ **then**
        for $i = 1;\ i \leq n;\ i = i + 1$ **do**
            $I_i = I_i\backslash\{u_i\}$
        end for
    else
        return I
    end if
end while
return $\emptyset$

end

The language we are learning can have a much exponentially larger alphabet than the RTS. However, *Libalf* - the software we are using - allows us to start the learning process with a smaller alphabet, which we can expand later if we need to. So, we begin with an empty alphabet and gradually add letters from 2^{Σ}.

6. Experiments

6.1. Case studies

Because the main work of this thesis is compare and evaluate performance between of algorithm, we just consider some simple case studies.

Dijkstra's algorithm for mutual exclusion with a token The example illustrates a mutual exclusion algorithm [FO97] for agents forming a ring. They pass around a single token as a semaphore for a critical region.

Other mutual exclusion algorithms Additionally, we also consider the mutual exclusion algorithms of the standard bakery algorithm [Che+17].

Dining philosophers Atomic

Cache coherence protocols When checking for cache coherence protocols, it's important to ensure that there aren't two different versions of the same data point present in the cache. In this regard, we only analyze the protocol MESI. We also examine different custom safety properties for each of these protocols. The models are babse on [Del00].

Leader election HERMAN

Token passing In conclusion, this thesis presents several examples that demonstrate token passing algorithms that we previously introduced in Chapter 2.

6.2. Dodo-cpp

Dodo-cpp is a program that has three interpretations, namely trap, siphon, and flow. It runs for every system with four algorithms, which are L^*, NL^*, Kearns and Vazirani, Rivest-Schapire. After that, it plotted the graphs to compare the learned time of theses algorithms.

After running the algorithms, Dodo-cpp plots graphs to compare the time taken for each algorithm to learn. To make the data easier to analyze, the program uses the mathplot library [Hun07] for graph plotting.

6.3. Graphs

The experiments in this thesis were running on x64 Ubuntu 22.04 system with 12th Gen Intel(R) Core(TM) i7-12700F processor and 16GiB memory.

For all Figures, we use KV as an abbreviation for Kearns and Vazirani's algorithm and RS for Rivest-Schapire's algorithm. All of the algorithms we used to learn produced the same results, regardless of their configuration and whether they could learn. However, the learning time for each algorithm varied. In the case of learning for a token-passing system using Trap and TrapSAT [1] interpretation and the "notoken" property, Figure 6.1 shows that the inductive statements can be learned (marked with blue bars). We can also compare the learning time between the algorithms to determine which one is more effective.

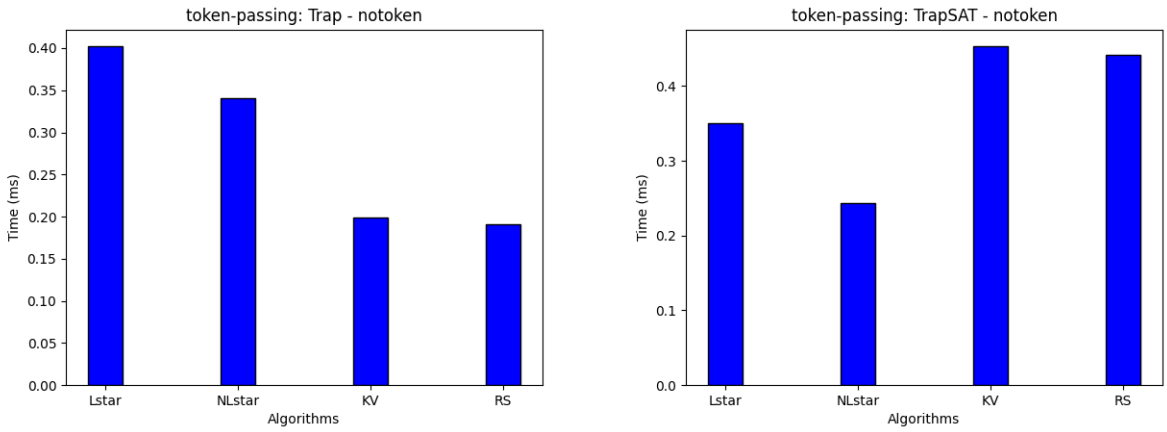


Figure 6.1.: *Comparing the learning time of all algorithms; Result: success; Interpretation: Trap, TrapSAT; System: token-passing; property: notoken.*

In Figure 6.2, we cannot learn using the Siphon and SiphonSAT[1] interpretation. We denote these cases with red bars.

[1] Use SAT-Sovler for the word problem.

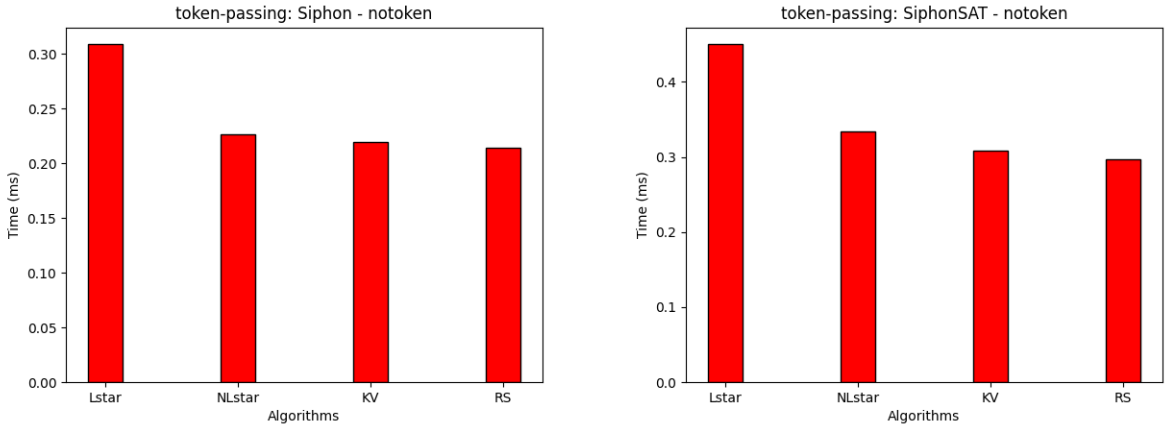


Figure 6.2.: *Comparing the learning time of all algorithms; Result: fail; Interpretation: Siphon, SiphonSAT; System: token-passing; property: notoken.*

6.4. Evaluating

In Figure 6.8, we compare the number of memeber ship queries that are asked by each algorithm. Because in our algorithm, we add the alphabet gradually and it causes membership queries of all algorithm are the same.

we can not say for which algorithm is the best for all cases but at least we know in specifics configuration which one is better.

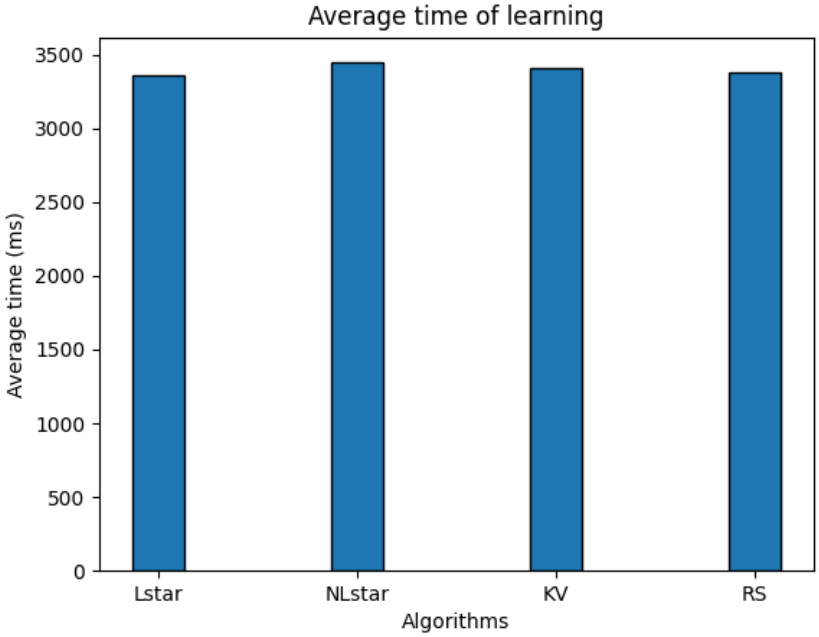


Figure 6.3.: *Comparing the average learning time of all algorithms.*

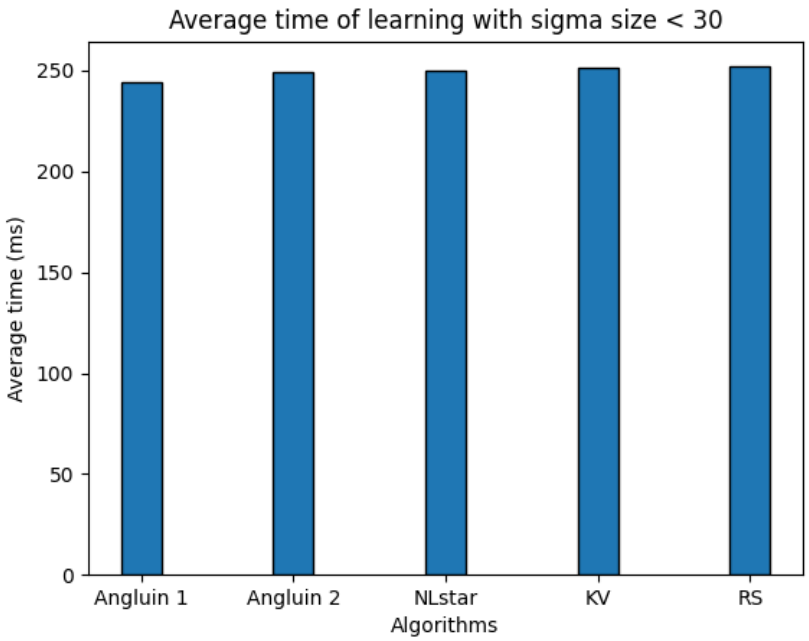


Figure 6.4.: *Comparing the average learning time of all algorithms for all configurations with have* $|\Sigma| < 30$.

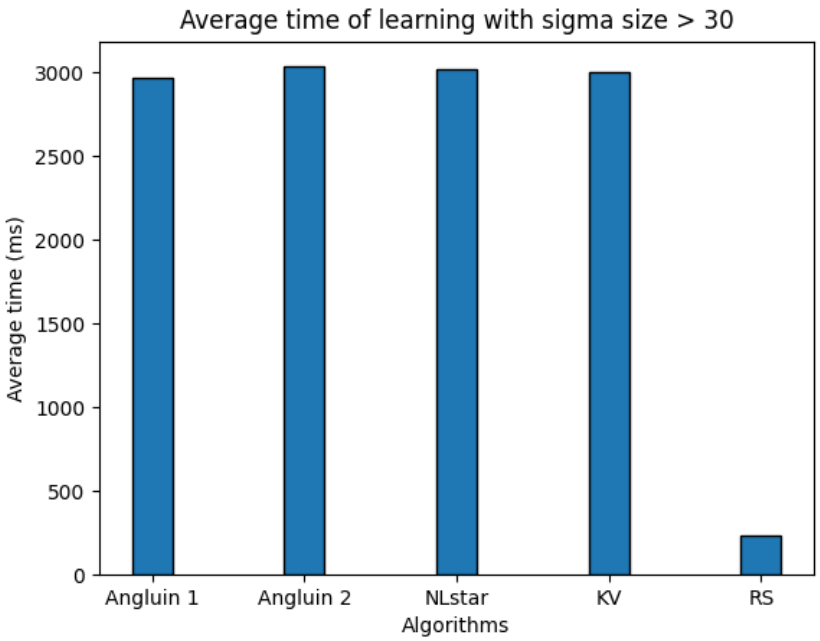


Figure 6.5.: *Comparing the average learning time of all algorithms for all configurations with have* $|\Sigma| \geq 30$.

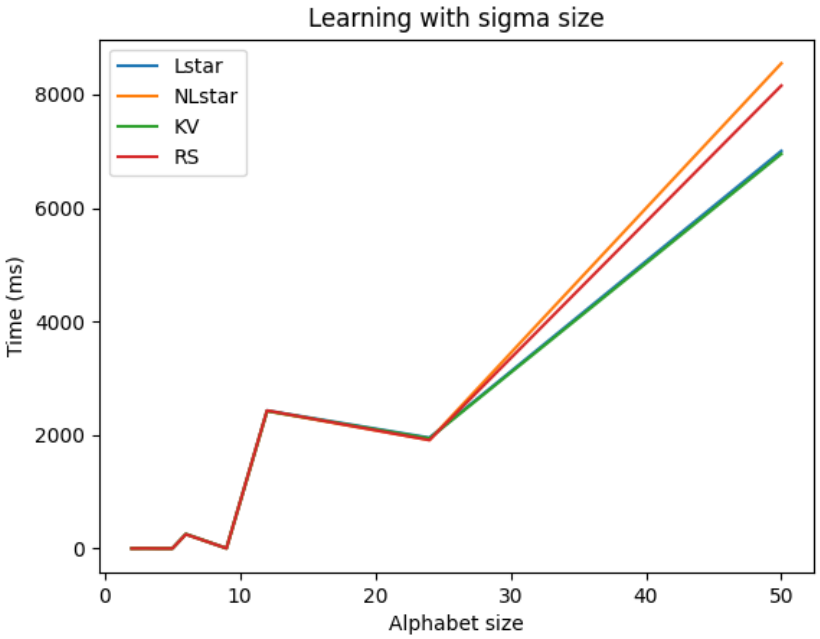


Figure 6.6.: *Comparing the average learning time of all algorithms corressponds to* $|\Sigma|$.

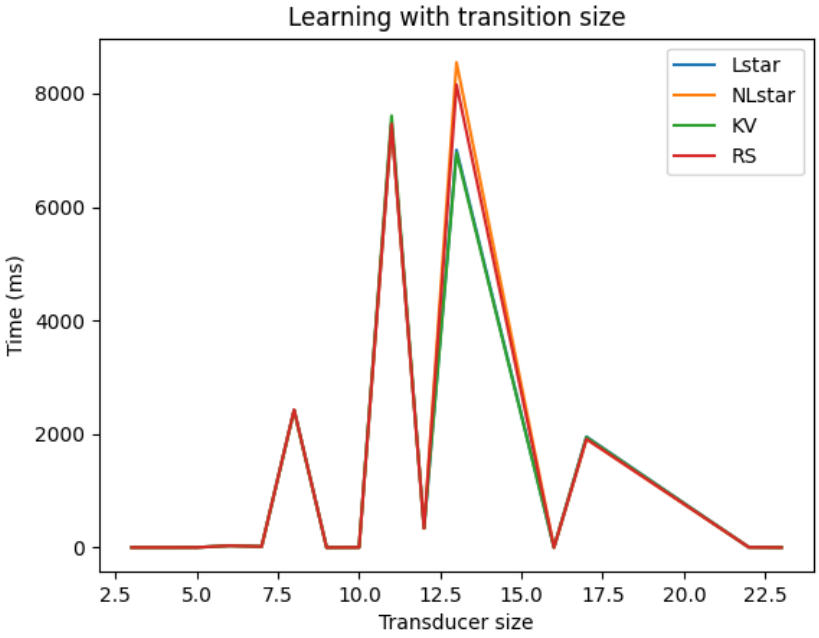


Figure 6.7.: *Comparing the average learning time of all algorithms corressponds to transducer size.*

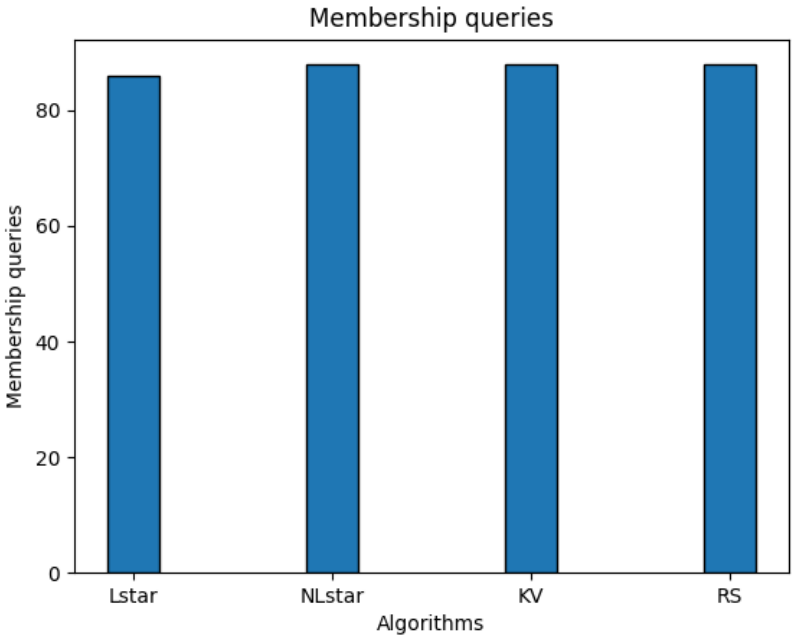


Figure 6.8.: *Comparing the average membership queries of all algorithms.*

7. Conclusion

We studied Regular Model Checking of safety properties. We evaluated the performance of our algorithms based on a prototype implementation.

NEED MORE INFO FROM EXPERIMENT

Open Questions and Future Research

NEED MORE INFO FROM EXPERIMENT

A. Experiments results

A.1. Dijkstra's algorithm for mutual exclusion with a token

A.2. Other mutual exclusion algorithms

Name	Property	Interpretations	Algorithms	Result	Time	Language Size
Berkeley	exclusiveexclusive	Trap	Lstar	x	2.472ms	-
			NLstar	x	2.547ms	-
			KV	x	2.77ms	-
			RS	x	2.254ms	-
		Siphon	Lstar	x	1.829ms	-
			NLstar	x	1.978ms	-
			KV	x	1.67ms	-
			RS	x	1.688ms	-
		Flow	Lstar	x	5.556ms	-
			NLstar	x	5.435ms	-
			KV	x	5.484ms	-
			RS	x	5.428ms	-

Table A.1.

Name	Property	Interpretations	Algorithms	Result	Time	Language Size
Berkeley	exclusiveunowned	Trap	Lstar	v	27.835ms	6
			NLstar	v	25.548ms	6
			KV	v	25.213ms	6
			RS	v	25.961ms	6
		Siphon	Lstar	x	1.19ms	-
			NLstar	x	1.121ms	-
			KV	x	1.091ms	-
			RS	x	1.091ms	-
		Flow	Lstar	x	5.627ms	-
			NLstar	x	5.916ms	-
			KV	x	5.636ms	-
			RS	x	5.861ms	-

Table A.2.

Name	Property	Interpretations	Algorithms	Result	Time	Language Size
Berkeley	exclusivenonexclusive	Trap	Lstar	v	33.441ms	6
			NLstar	v	31.765ms	6
			KV	v	30.473ms	6
			RS	v	30.338ms	6
		Siphon	Lstar	x	1.953ms	-
			NLstar	x	1.74ms	-
			KV	x	1.682ms	-
			RS	x	1.664ms	-
		Flow	Lstar	x	5.616ms	-
			NLstar	x	5.48ms	-
			KV	x	5.422ms	-
			RS	x	5.85ms	-

Table A.3.

Name	Property	Interpretations	Algorithms	Result	Time	Language Size
Berkeley	deadlock	Trap	Lstar	v	0.219ms	1
			NLstar	v	0.094ms	1
			KV	v	0.09ms	1
			RS	v	0.079ms	1
		Siphon	Lstar	v	0.105ms	1
			NLstar	v	0.089ms	1
			KV	v	0.08ms	1
			RS	v	0.078ms	1
		Flow	Lstar	v	0.118ms	1
			NLstar	v	0.083ms	1
			KV	v	0.08ms	1
			RS	v	0.078ms	1

Table A.4.

A.3. Dining philosophers

A.4. Cache coherence protocols

A.5. Leader election

A.6. Token passing

List of Figures

List of Algorithms

List of Tables

Bibliography

[Abd+04] P. A. Abdulla, B. Jonsson, M. Nilsson, and M. Saksena. "A Survey of Regular Model Checking." In: *CONCUR 2004 - Concurrency Theory*. Ed. by P. Gardner and N. Yoshida. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 35–48.

[AMP22] B. K. Aichernig, E. Muškardin, and A. Pferscher. "Active vs. passive: a comparison of automata learning paradigms for network protocols." In: *arXiv preprint arXiv:2209.14031* (2022).

[Ang87] D. Angluin. "Learning regular sets from queries and counterexamples." In: *Information and Computation* 75.2 (1987), pp. 87–106. ISSN: 0890-5401. DOI: https://doi.org/10.1016/0890-5401(87)90052-6.

[BFL04] S. Bardin, A. Finkel, and J. Leroux. "FASTer Acceleration of Counter Automata in Practice." In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by K. Jensen and A. Podelski. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 576–590.

[Blo+16] R. Bloem, S. Jacobs, A. Khalimov, I. Konnov, S. Rubin, H. Veith, and J. Widder. "Decidability in parameterized verification." In: *ACM SIGACT News* 47.2 (2016), pp. 53–64.

[Bol+09] B. Bollig, P. Habermehl, C. Kern, and M. Leucker. "Angluin-Style Learning of NFA." In: *International Joint Conference on Artificial Intelligence*. 2009.

[Bol+10] B. Bollig, J.-P. Katoen, C. Kern, M. Leucker, D. Neider, and D. R. Piegdon. "libalf: The automata learning framework." In: *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*. Springer. 2010, pp. 360–364.

[Bou+00] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. "Regular model checking." In: *Computer Aided Verification: 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000. Proceedings 12*. Springer. 2000, pp. 403–418.

[Che+17] Y.-F. Chen, C.-D. Hong, A. W. Lin, and P. Ruemmer. *Learning to Prove Safety over Parameterised Concurrent Systems (Full Version)*. 2017. arXiv: 1709.07139 [cs.LO].

[De 05] C. De La Higuera. "A bibliographical study of grammatical inference." In: *Pattern recognition* 38.9 (2005), pp. 1332–1348.

[Del00] G. Delzanno. "Automatic verification of parameterized cache coherence protocols." In: *Computer Aided Verification: 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000. Proceedings 12*. Springer. 2000, pp. 53–68.

[DLT01] F. Denis, A. Lemay, and A. Terlutte. "Residual Finite State Automata." In: *STACS 2001*. Ed. by A. Ferreira and H. Reichel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 144–157. ISBN: 978-3-540-44693-4.

[FO97] L. Fribourg and H. Olsén. "Reachability sets of parameterized rings as regular languages." In: *Electronic Notes in Theoretical Computer Science* 9 (1997), p. 40.

[Hol+15] A. Holzer, C. Schallhart, M. Tautschnig, and H. Veith. "Closure properties and complexity of rational sets of regular languages." In: *Theoretical Computer Science* 605 (2015), pp. 62–79.

[Hun07] J. D. Hunter. "Matplotlib: A 2D graphics environment." In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[Kni] S. Knieschewski. "Berühmt berüchtigte Softwarefehler: Der Pentium Division-Bug." In.

[Koo14] P. Koopman. "A case study of Toyota unintended acceleration and software safety." In: *Presentation. Sept* (2014).

[KV94] M. J. Kearns and U. Vazirani. *An introduction to computational learning theory*. MIT press, 1994.

[Nei14] D. Neider. "Applications of automata learning in verification and synthesis." In: 2014.

[Ozo+05] R. Ozols, R. Freivalds, L. Mancinska, and M. Ozols. "Size of Nondeterministic and Deterministic Automata for Certain Languages." In: *FCS*. 2005, pp. 169–175.

[RS89] R. L. Rivest and R. E. Schapire. "Inference of finite automata using homing sequences." In: *Proceedings of the twenty-first annual ACM symposium on Theory of computing*. 1989, pp. 411–420.

[Wel23] C. Welzel-Mohr. "Inductive Statements for Regular Transition Systems." In: 2023.

[Wik23] Wikipedia contributors. *De Morgan's laws — Wikipedia, The Free Encyclopedia*. [Online; accessed 15-December-2023]. 2023.